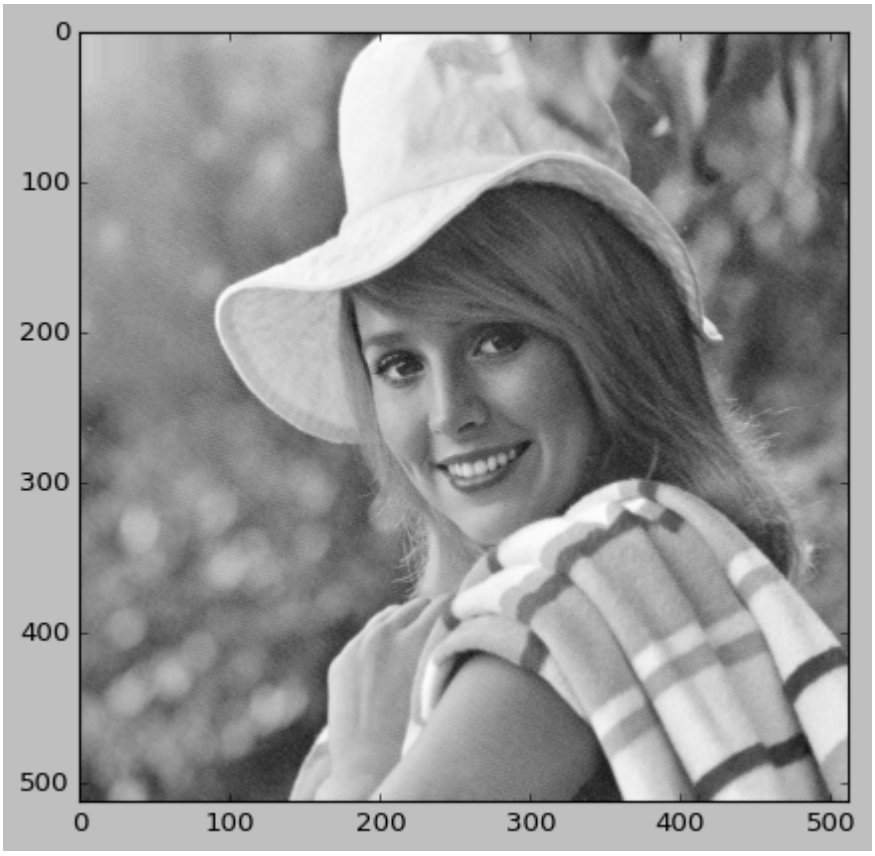
In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import cv2
from functions import *

plt.style.use(plt.style.available[5])
```

3.1. Box Filter

3.1.3. What is the resulting filter if apply the 3 × 3 box filter many times? (Test on grayscale Elaine Image).

In [ ]:
```
elaine = cv2.imread('Elaine.bmp',cv2.IMREAD_GRAYSCALE)
plt.imshow(elaine,cmap='gray')
print(elaine.dtype)

figure = plt.figure(figsize=(19,19))

for i in range(16):
    elaine_padded = clip_filter(elaine,1)
    elaine = box_filter(elaine_padded,3,1)

    figure.add_subplot(4,4,i+1)
    plt.imshow(elaine,cmap='gray')
    plt.title(f"iteration({i+1})",color='black')
```

```
uint8
```

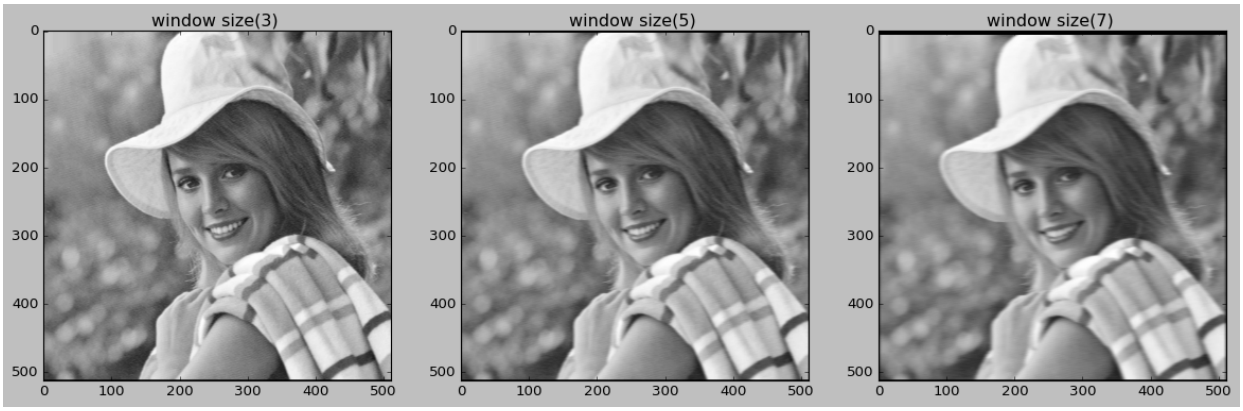


3.1.4. How does the size of the mask affect blurring and noise reduction? (Test on grayscale Elaine Image).

In [ ]:
```
elaine = cv2.imread('Elaine.bmp',cv2.IMREAD_GRAYSCALE)

figure = plt.figure(figsize=(19,19))

for i in range(12):
    # elaine_padded = clip_filter(elaine,3)
    elaine = box_filter(elaine)
    figure.add_subplot(3,4,i+1)
    plt.imshow(elaine,cmap='gray')
    plt.title(f"iteration({i+1})",color='black')
```

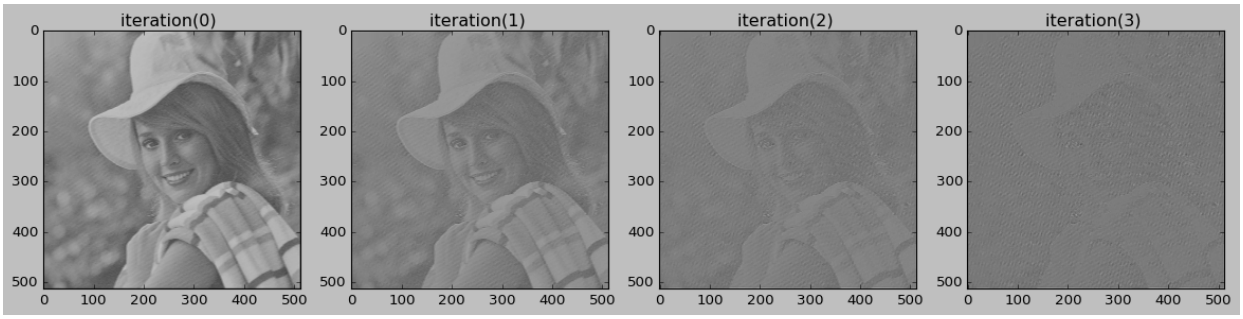
3.1.5. Which mask size do you think provides a better tradeoff between blurring and noise reduction for this image?

In [ ]:
```
elaine = cv2.imread('Elaine.bmp',cv2.IMREAD_GRAYSCALE)

figure = plt.figure(figsize=(18,18))
n = 1 # padding size
m = 3 # window size
for i in range(3):

    elaine_padded = clip_filter(elaine,n)
    elaine = box_filter(elaine_padded,m,n)

    figure.add_subplot(3,3,i+1)
    plt.imshow(elaine,cmap='gray')
    plt.title(f"window size({m}) ",color='black')

    n +=1
    m +=2

plt.show()
```



3.1.6. What is the resulting if apply Laplacian mask ([0 -1 0; -1 5 -1;0 -1 0]) many times? (Test output of 3.1.3).

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import cv2
from functions import *

plt.style.use(plt.style.available[5])
elaine = cv2.imread('Elaine.bmp',cv2.IMREAD_GRAYSCALE)
elaine_padded = clip_filter(elaine,2)
box_elaine = box_filter(elaine_padded,5,2)
mask = np.array([0,-1,0,-1,5,-1,0,-1,0])
mask = mask.reshape(3,3)
figure = plt.figure(figsize=(18,18))

for i in range(4):
    box_elaine = laplacian_filter(box_elaine,mask)

    figure.add_subplot(1,4,i+1)
    plt.imshow(box_elaine,cmap='gray')
    plt.title(f"iteration({i}) ",color='black')
```



3.2. Median Filter

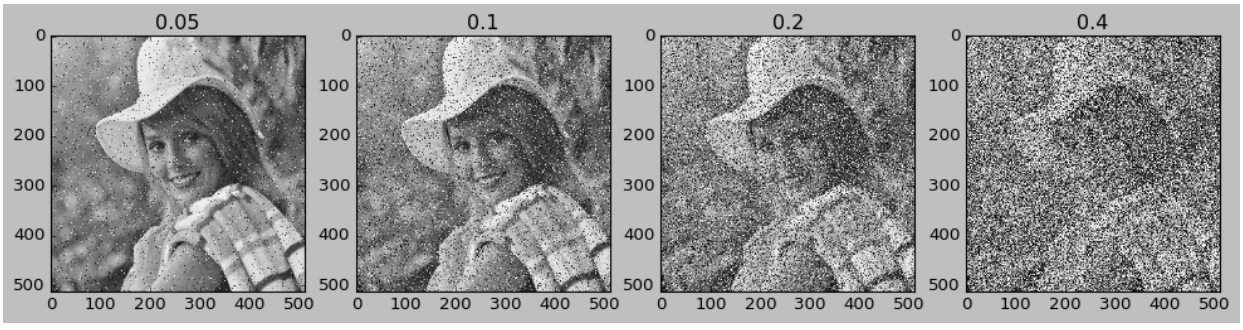
3.2.1. Write a program that can, first, add salt-and-pepper noise to an image with a specified noise density. Try different noise density (0.05, 0.1, 0.2, 0.4). Then, perform median filtering with a specified window size. Consider only the median filter with a square shape. For each density, discuss the effect of filtering with different window sizes (3, 5, 7, 9) and experimentally determine the best window size. Note: you can use imnoise and immse functions to generate noisy images and compare the quality of images, respectively. Also, you can ignore the

boundary problem by only performing the filtering for the pixels inside the boundary. (Test on grayscale Elaine Image).

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import cv2
from functions import *

plt.style.use(plt.style.available[5])
```

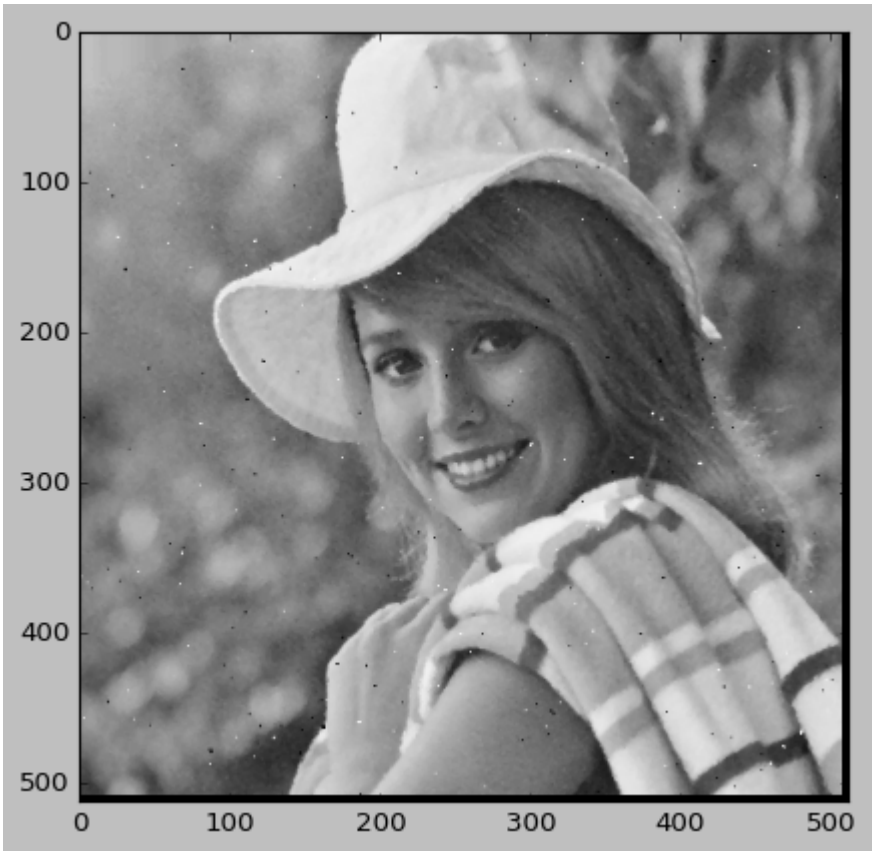
In [ ]:
```
eliane = cv2.imread('Elaine.bmp',cv2.IMREAD_GRAYSCALE)


figure = plt.figure(figsize=(15,15))

elaine_noisey = salt_pepper(eliane,0.05)
figure.add_subplot(1,4,1)
plt.imshow(elaine_noisey,cmap='gray')
plt.title('0.05')

elaine_noisey = salt_pepper(eliane,0.1)
figure.add_subplot(1,4,2)
plt.imshow(elaine_noisey,cmap='gray')
plt.title('0.1')

elaine_noisey = salt_pepper(eliane,0.2)
figure.add_subplot(1,4,3)
plt.imshow(elaine_noisey,cmap='gray')
plt.title('0.2')

elaine_noisey = salt_pepper(eliane,0.4)
figure.add_subplot(1,4,4)
plt.imshow(elaine_noisey,cmap='gray')
plt.title('0.4')
```

Out[ ]:
```
Text(0.5, 1.0, '0.4')
```



In [ ]:
```
eliane = cv2.imread('Elaine.bmp',cv2.IMREAD_GRAYSCALE)
elaine_noisey = salt_pepper(eliane,0.2)
elaine_median = median_filter(eliane,windowSize=3)
plt.imshow(elaine_median,cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x1d7fe652b00>
```



In [ ]:
```
elaine = cv2.imread('Elaine.bmp',cv2.IMREAD_GRAYSCALE)
figure =  plt.figure(figsize=(18,18))
noise_density = [0.05 , 0.1, 0.2, 0.4]
windowSizes = [3,5,7,9]
for i,nd in enumerate(noise_density):
    for j,ws in enumerate(windowSizes):
        elaine_noisey = salt_pepper(elaine,nd)
        elaine_median = median_filter(eliane,windowSize=ws)
        # calculate MSE, for original image cases
        mse = mean_square_error(elaine,elaine_median)
        print(f'mean_square_error({i,nd}) = {mse}')
        figure.add_subplot(4,4,(4*i)+(j+1))
        plt.imshow(elaine_median,cmap='gray')
        plt.title(f'noise density:{nd}\nwindow size:{ws}')
```

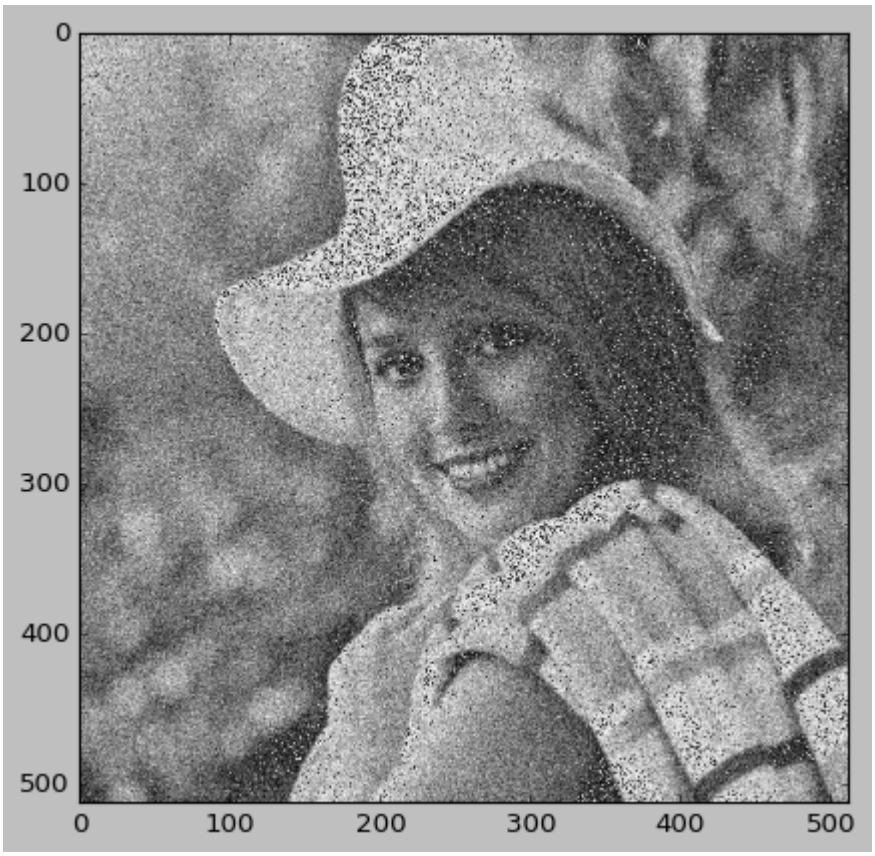
```
mean_square_error((0, 0.05)) = 1318.4843
mean_square_error((0, 0.05)) = 2420.4907
mean_square_error((0, 0.05)) = 3532.5902
mean_square_error((0, 0.05)) = 4578.3781
mean_square_error((1, 0.1)) = 5487.9209
mean_square_error((1, 0.1)) = 7140.6039
mean_square_error((1, 0.1)) = 8686.6320
mean_square_error((1, 0.1)) = 10105.6397
mean_square_error((2, 0.2)) = 11786.5115
mean_square_error((2, 0.2)) = 13721.0691
mean_square_error((2, 0.2)) = 15205.5746
mean_square_error((2, 0.2)) = 16359.9473
mean_square_error((3, 0.4)) = 17408.0267
mean_square_error((3, 0.4)) = 18146.7812
mean_square_error((3, 0.4)) = 18561.2295
mean_square_error((3, 0.4)) = 18776.5542
```

3.2.2. Create a program for adding Gaussian noise with different variance and filtering using average and median filter, respectively. Apply the averaging filter and the median filter to an image with Gaussian noise (with a chosen noise variance). Discuss the effectiveness of each filter on this type of noise. Note: You can use imnoise and immse functions to generate noisy images and compare the quality of images, respectively. (Test on grayscale Elaine Image).

In [ ]:

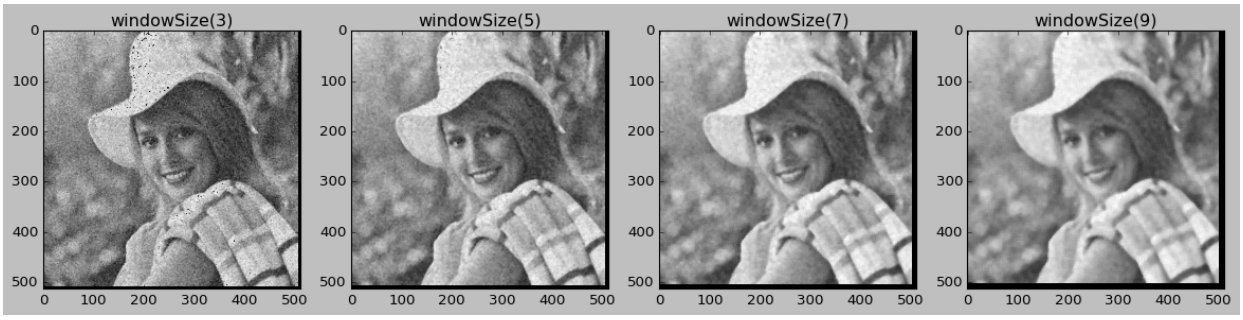
```
import numpy as np
import cv2
from matplotlib import pyplot as plt
elaine = cv2.imread("Elaine.bmp",cv2.IMREAD_GRAYSCALE)
mean = 0
var = 80
sigma = var ** 0.8
gaussian = np.random.normal(mean, sigma, size=elaine.shape)
noisy_image = np.zeros_like(elaine)
noisy_image = np.add(elaine,gaussian.astype('uint8'))
plt.imshow(noisy_image, cmap='gray')
```

Out[ ]: <matplotlib.image.AxesImage at 0x1d7fcc7feb0>



In [ ]:
```
figure = plt.figure(figsize=(18,18))
window = [3,5,7,9]

for i in range(4):
    elaine_gaussin_medain = median_filter(noisy_image,window[i])
    print(mean_square_error(elaine,elaine_gaussin_medain))
    figure.add_subplot(1,4,i+1)
    plt.imshow(elaine_gaussin_medain,cmap='gray')
    plt.title(f"windowSize({window[i]})",color='black')
```

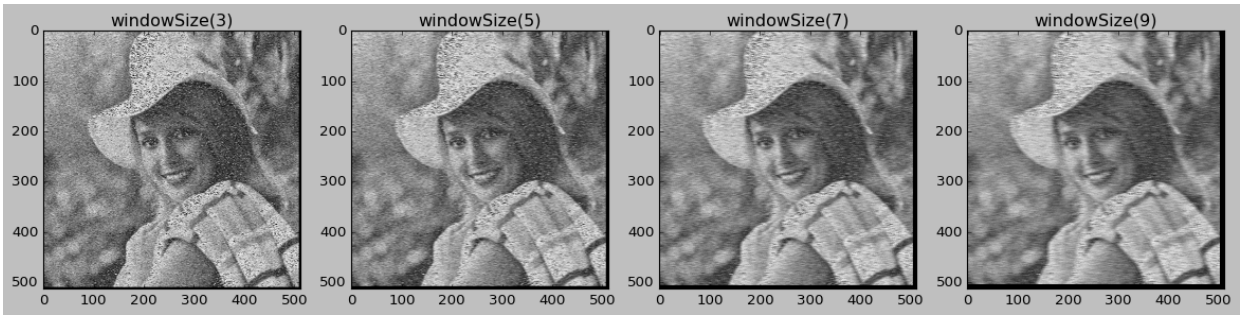
```
632.2918
709.8487
927.0042
1169.1656
```



In [ ]:
```
figure = plt.figure(figsize=(18,18))
window = [3,5,7,9]
for i in range(4):
    elaine_gaussin_box = box_filter(noisy_image,window[i])
    print(mean_square_error(elaine,elaine_gaussin_box))
    figure.add_subplot(1,4,i+1)
```

```
    plt.imshow(elaine_gaussin_box,cmap='gray')
    plt.title(f"windowSize({window[i]})",color='black')
```

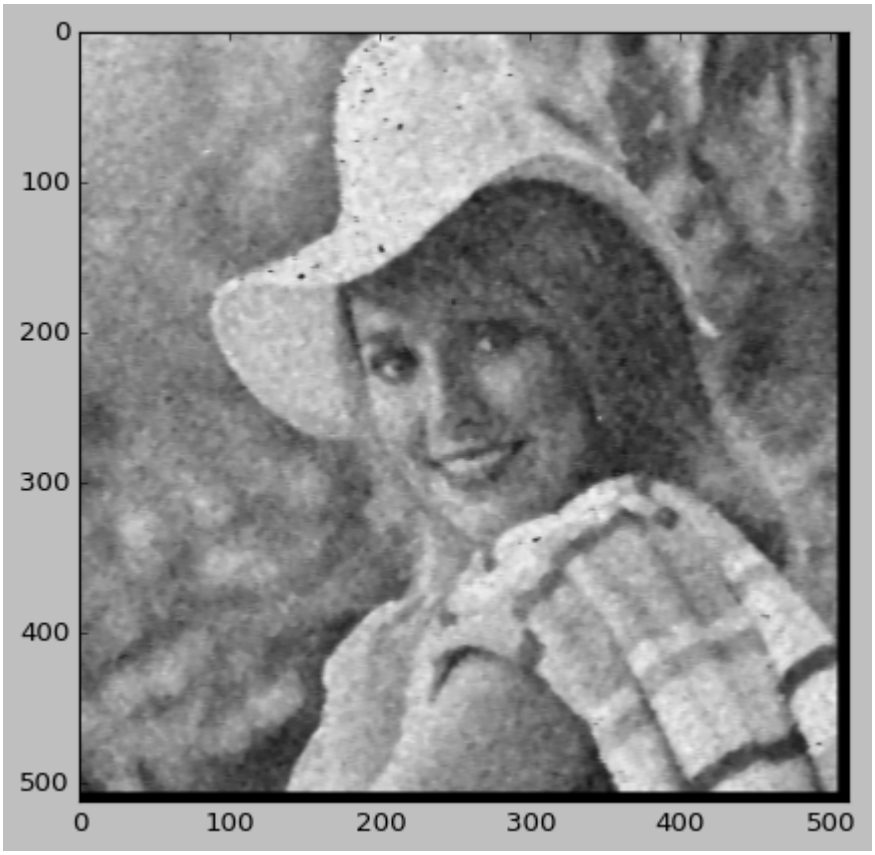
```
1004.6007
977.1495
1079.5323
1225.8927
```



3.2.3. Design proper filters to simultaneously remove Gaussian and salt & pepper noise. Please detail the steps of the denoising process and specify corresponding parameters. Provide some discussions about the reason why those filters and parameters are chosen.

In [ ]:
```
elaine = cv2.imread("Elaine.bmp" , cv2.IMREAD_GRAYSCALE)
mean = 0
var = 70
sigma = var ** 0.8
gaussian = np.random.normal(mean, sigma, size=elaine.shape)
noisy_image = np.zeros_like(elaine)
noisy_image = np.add(elaine,gaussian.astype('uint8'))

noisy_image2 = salt_pepper(noisy_image,0.3)
elaine1 = median_filter(noisy_image2,windowSize=5)
elaine2 = box_filter(elaine1,windowSize=3)

plt.imshow(elaine2,cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x1d7fcbe4bb0>
```

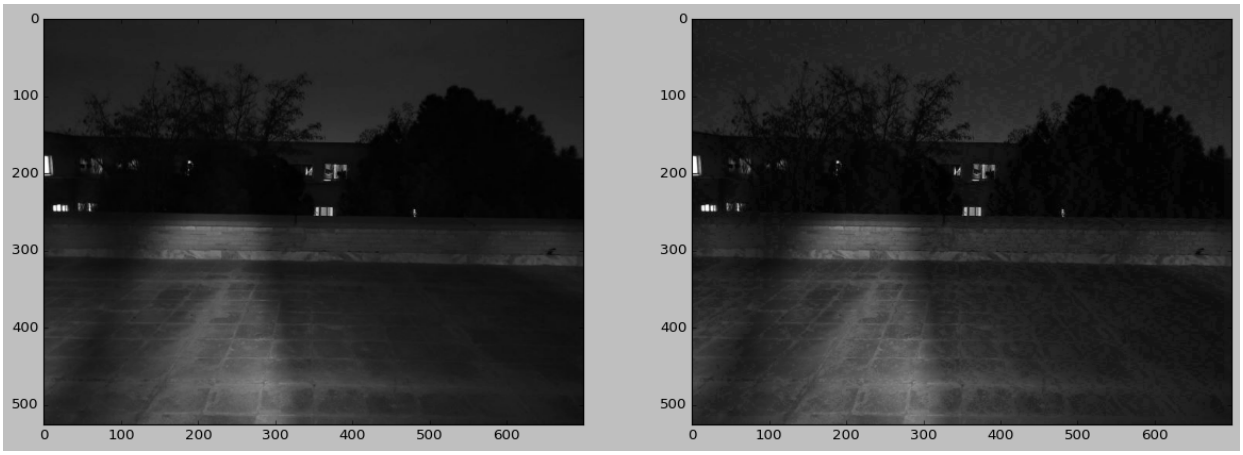


3.3. Sharpening, Blurring, and Noise Removal

3.3.1. Take blurry and noisy images (shooting in low light is a good way to get both) by your cellphone and try to improve their appearance and legibility. Display and discuss the results before and after improving.

In [ ]:
```
sample = cv2.imread('sample3.jpg',cv2.IMREAD_GRAYSCALE)

figure = plt.figure(figsize=(18,18))
figure.add_subplot(1,2,1)
plt.imshow(sample,cmap='gray')

sample_gaussin = box_filter(sample,windowSize=11)
image2 = sample_gaussin - sample
a = 0.020
image3 = np.multiply(image2,a)
image4 = sample + image3
figure.add_subplot(1,2,2)
plt.imshow(image4,cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x1d7fdfbc5b0>
```



3.4. Edge Detection

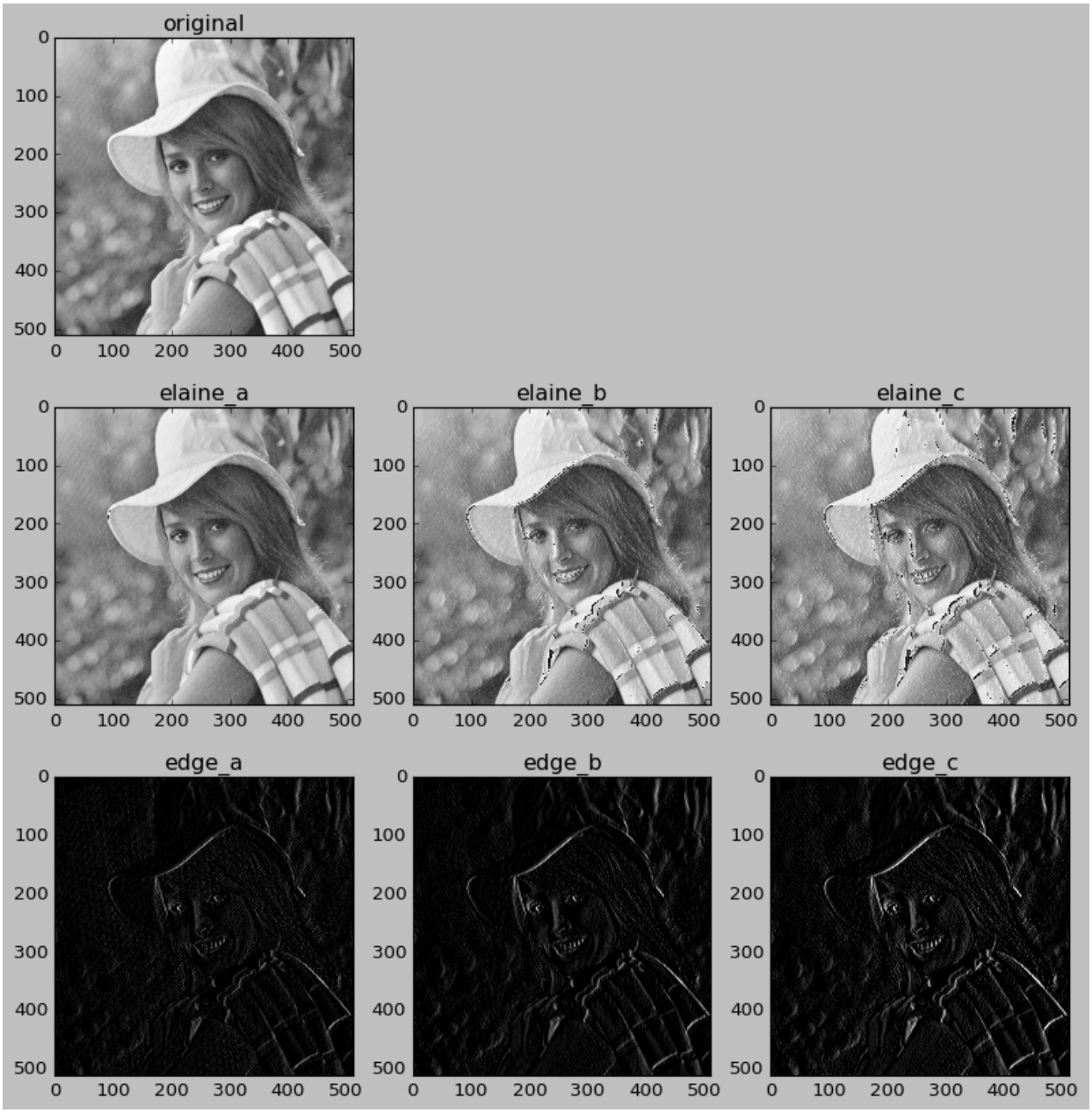
3.4.1. These are often used first-order difference filters in x-direction: a) 1/2 [1 0 −1], b) 1/6 [1 0 −1 1 0 −1 1 0 −1], c) 1/8 [1 0 −1 2 0 −2 1 0 −1] Compare and describe the properties of the three filters; In the following discuss the efficiency of filters for edge detection problems. Are these filters suitable for computing the 2-D gradient? (Test on Elaine Lena Image)

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
from functions import *

plt.style.use(plt.style.available[5])


elaine = cv2.imread('Elaine.bmp',cv2.IMREAD_GRAYSCALE)

mask1 = np.array([1,0,-1])
mask1 = mask1.reshape(1,3)
mask2 = np.array([1,0,-1,1,0,-1,1,0,-1])
mask2 = mask2.reshape(3,3)
mask3 = np.array([1,0,-1,2,0,-2,1,0,-1])
mask3 = mask3.reshape(3,3)

elaine_a , filter_a = weighted_filter(elaine,mask1,(0.5))
elaine_b , filter_b = weighted_filter(elaine,mask2,(1/6))
elaine_c , filter_c = weighted_filter(elaine,mask3,(1/8))

figure = plt.figure(figsize=(12,12))

figure.add_subplot(3,3,1)
plt.imshow(elaine,cmap='gray')
plt.title('original')

figure.add_subplot(3,3,4)
plt.imshow(elaine_a,cmap='gray')
plt.title('elaine_a')

figure.add_subplot(3,3,5)
plt.imshow(elaine_b,cmap='gray')
plt.title('elaine_b')

figure.add_subplot(3,3,6)
```

```
plt.imshow(elaine_c,cmap='gray')
plt.title('elaine_c')

figure.add_subplot(3,3,7)
plt.imshow(filter_a,cmap='gray')
plt.title('edge_a')

figure.add_subplot(3,3,8)
plt.imshow(filter_b,cmap='gray')
plt.title('edge_b')

figure.add_subplot(3,3,9)
plt.imshow(filter_c,cmap='gray')
plt.title('edge_c')
```

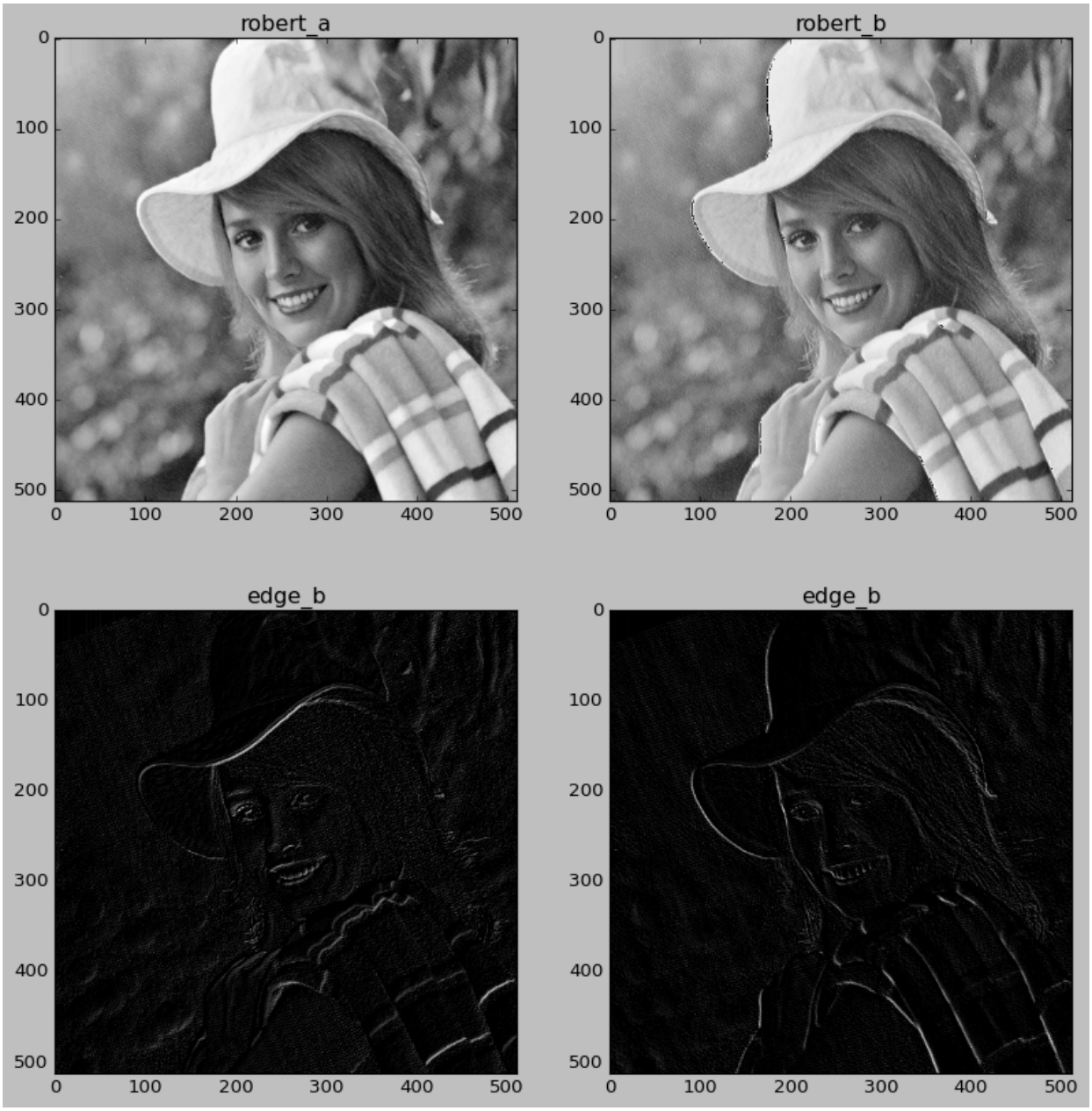
Out[ ]: Text(0.5, 1.0, 'edge_c')



3.4.2. Robert suggested the filter to compute the 2-D gradient and to detect edges. a) [1 0 0 −1], b) [0 1 −1 0] First, in which directions do these filters detect edges? Second, compare the quality of this filter with the filter from previous steps. (Test on grayscale Elaine Image).

In [ ]:
```
elaine = cv2.imread('Elaine.bmp',cv2.IMREAD_GRAYSCALE)
mask1 = np.array([1,0,0,-1])
mask1 = mask1.reshape(2,2)
mask2 = np.array([0,1,-1,0])
mask2 = mask2.reshape(2,2)

robert_a , filter_a = weighted_filter(elaine,mask1)
robert_b , filter_b = weighted_filter(elaine,mask2)
# elaine_c = weighted_filter(elaine,mask3,(1/8))

figure = plt.figure(figsize=(12,12))

figure.add_subplot(2,2,1)
plt.imshow(robert_a,cmap='gray')
plt.title('robert_a')

figure.add_subplot(2,2,2)
plt.imshow(robert_b,cmap='gray')
plt.title('robert_b')

figure.add_subplot(2,2,3)
plt.imshow(filter_a,cmap='gray')
plt.title('edge_b')

figure.add_subplot(2,2,4)
plt.imshow(filter_b,cmap='gray')
plt.title('edge_b')
```

Out[ ]:
```
Text(0.5, 1.0, 'edge_b')
```



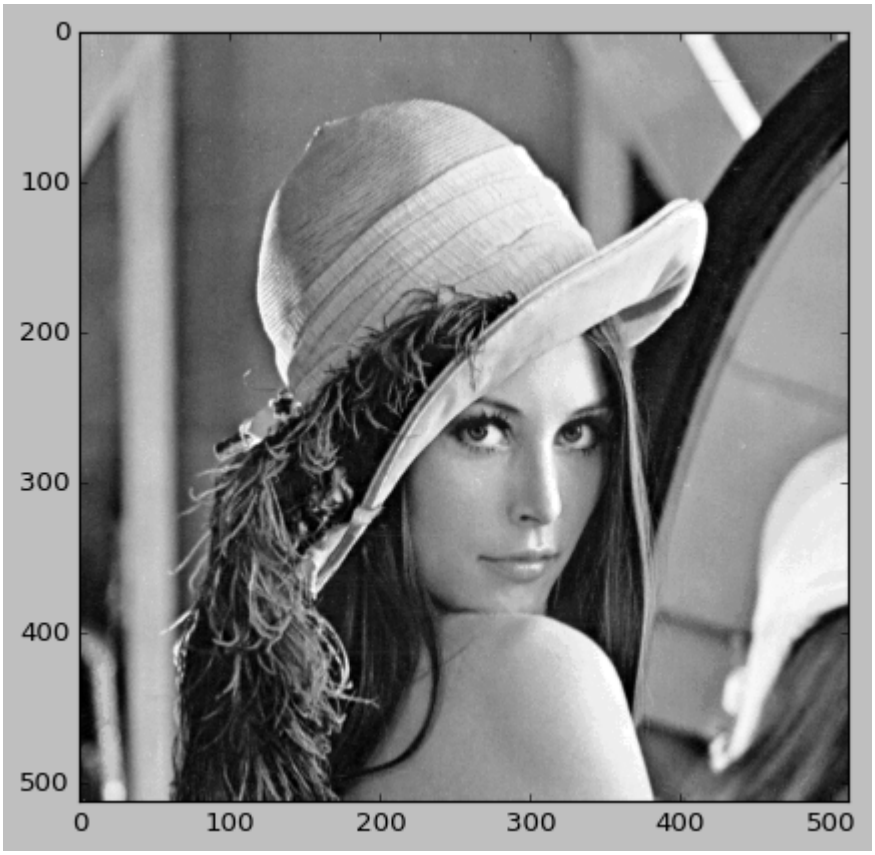
3.5. Unsharp Masking

3.5.1. A simple unsharp masking filter has the following form: $(1 - \alpha)I + \alpha I' = I + \alpha(I' - I)$, where $\alpha \in [0, 1]$ represents the threshold step of strength sharpening and $I'$is smoothed version of I which obtained using different filter sizes based on the gaussian filter. Compare and discuss the results for different filter sizes (3, 5, 7, 9, 11) and α for the grayscale Lena image. What happens if we set α very large or small? How to obtain the optimum value? Note: You can use imgaussfilt function for smoothing.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import cv2
from functions import *
# from skimage import
plt.style.use(plt.style.available[5])
```

In [ ]:

```
lena = cv2.imread('Lena.bmp',cv2.IMREAD_GRAYSCALE)
plt.imshow(lena,cmap='gray')
figure = plt.figure(figsize=(18,23))
windowSizes = [3,5,7,9,11]
density = [0.02,0.1,0.25,0.5]
for i , ws in enumerate(windowSizes):
    for j , nd in enumerate(density):
        lena = cv2.imread('Lena.bmp',cv2.IMREAD_GRAYSCALE)
        lena_gaussin = box_filter(lena,windowSize=windowSizes[i])
        image2 = lena_gaussin - lena
        a = density[j]
        image3 = np.multiply(image2,a)
        image4 = lena + image3
        figure.add_subplot(5,4,(4*i)+(j+1))
        plt.imshow(image4,cmap='gray')
        plt.title(f'widowSize({windowSizes[i]})\ndensity({density[j]})')
```




widowSize(3)
density(0.02)
widowSize(3)
density(0.1)
widowSize(3)
density(0.25)
widowSize(3)
density(0.5)
widowSize(5)
density(0.02)
widowSize(5)
density(0.1)
widowSize(5)
density(0.25)
widowSize(5)
density(0.5)
widowSize(7)
density(0.02)
widowSize(7)
density(0.1)
widowSize(7)
density(0.25)
widowSize(7)
density(0.5)
widowSize(9)
density(0.02)
widowSize(9)
density(0.1)
widowSize(9)
density(0.25)
widowSize(9)
density(0.5)
widowSize(11)
density(0.02)
widowSize(11)
density(0.1)
widowSize(11)
density(0.25)
widowSize(11)
density(0.5)
0
100
200
300
400
500